

IMEX Methods in SUNDIALS: The ARKode Solver

Daniel R. Reynolds

`reynolds@smu.edu`

Department of Mathematics
Southern Methodist University

BOUT++ 2013 Workshop
September 4, 2013

Outline

- 1 Motivation
- 2 ARK Methods
- 3 Example Results
- 4 Conclusions

Outline

- 1 Motivation
- 2 ARK Methods
- 3 Example Results
- 4 Conclusions

Multiphysics Problems

“Multiphysics problems” typically involve a variety of interacting processes:

- System of multiple components coupled in the bulk:
 - Cosmology: radiation + (magneto)hydrodynamics + chemistry + gravity
 - Combustion/subsurface flow: reaction + transport
- System of multiple components coupled across interfaces:
 - Climate: ocean + atmosphere + sea ice
 - Tokamak: fluid core + kinetic edge

In this talk, we'll consider the prototypical multiphysics form,

$$\partial_t u = f(t, u) = f_1(t, u) + f_2(t, u),$$

where u comprises all of our unknowns, and $f_i(t, u)$ are different physical processes that each act on all or part of u .

A primary difficulty with multiphysics problems is that each process may evolve at a different speed, e.g. $f_1(t, u)$ is “slow” while $f_2(t, u)$ is “fast”.



Multiphysics Often Means “Multirate”

A single time scale is ideal for explicit-time methods, allowing for simpler algorithms, high-order accuracy, and predictable stability.

Wide temporal disparities can be analytically reformulated, but only for the scale of interest.

True multi-rate problems, however, require something more:

- Fully implicit methods are valid for stiff problems, but may require adaptation of solvers for all physical components.
- Operator-split methods are often chosen to match methods with physics.

Unfortunately, “standard” splitting approaches suffer from:

- *Low Accuracy* – even fractional-step methods may not achieve asymptotic $\mathcal{O}(h^2)$ accuracy until h is very small, since error terms are dominated by inter-process interactions [Ropp, Shadid,& Ober 2005].
- *Low Stability* – numerical stability isn't guaranteed *even if h is stable for each component* [Estep et al., 2007].



Fixing Operator Splitting

Although potentially dangerous, splittings are pervasive in scientific computing:

- Reuse existing/legacy software,
- Allow incorporation of domain-specific knowledge,
- No monolithic solvers for complex (and often non-differentiable) physics,
- Results “look reasonable,” especially once the time stepping parameters have been tweaked.

Can we enhance splitting’s stability & accuracy while retaining these benefits?

- The primary problem with basic splittings is that the component solvers are derived in isolation, with no concern for the coupling error.
- What if we instead derived new splitting approaches that explicitly account for inter-component coupling?

Outline

- 1 Motivation
- 2 ARK Methods**
- 3 Example Results
- 4 Conclusions

Additive Runge-Kutta Methods [Ascher et al. 1997; Araújo et al. 1997]

Although ARK methods may be derived for arbitrary splittings, here we consider splittings into two components: *explicit* and *implicit*,

$$\partial_t u = f_E(t, u) + f_I(t, u), \quad t \in [0, T], \quad u(0) = u_0,$$

We combine two s -stage methods (ERK + DIRK). Denoting $t_{n,j} = t_n + c_j h$,

$$z_i = u_n + h \sum_{j=1}^{i-1} a_{i,j}^E f_E(t_{n,j}, z_j) + h \sum_{j=1}^i a_{i,j}^I f_I(t_{n,j}, z_j), \quad i = 1, \dots, s,$$

$$u_{n+1} = u_n + h \sum_{j=1}^s b_j (f_E(t_{n,j}, z_j) + f_I(t_{n,j}, z_j)) \quad [\text{solution}]$$

$$\tilde{u}_{n+1} = u_n + h \sum_{j=1}^s \tilde{b}_j^E (f_E(t_{n,j}, z_j) + f_I(t_{n,j}, z_j)) \quad [\text{embedding}]$$

We therefore have two Butcher tables to work with:

| | | | | |
|----------|---------------|----------|-------------------|---------------|
| c_1 | 0 | 0 | \cdots | 0 |
| c_2 | $a_{2,1}^E$ | 0 | \cdots | 0 |
| \vdots | \vdots | \ddots | \ddots | \vdots |
| c_s | $a_{s,1}^E$ | \cdots | $a_{s,s-1}^E$ | 0 |
| | b_1 | \cdots | b_{s-1} | b_s |
| | \tilde{b}_1 | \cdots | \tilde{b}_{s-1} | \tilde{b}_s |

| | | | | |
|----------|---------------|-------------|-------------------|---------------|
| c_1 | $a_{1,1}^I$ | 0 | \cdots | 0 |
| c_2 | $a_{2,1}^I$ | $a_{2,2}^I$ | \cdots | 0 |
| \vdots | \vdots | \ddots | \ddots | \vdots |
| c_s | $a_{s,1}^I$ | \cdots | $a_{s,s-1}^I$ | $a_{s,s}^I$ |
| | b_1 | \cdots | b_{s-1} | b_s |
| | \tilde{b}_1 | \cdots | \tilde{b}_{s-1} | \tilde{b}_s |



ARK Coefficients

We have $s^2 + 3s$ total free parameters $(c_i, b_j, \tilde{b}_j, a_{i,j}^E, a_{i,j}^I)$. As with traditional RK methods, these are chosen to satisfy desired constraints:

- Maximize the order of accuracy for each elemental method,
- Maximize the stability of each elemental method,
- Simplify repeated implicit solves (e.g. SDIRK or ESDIRK),
- Enable accurate/stable embeddings for temporal error estimation,
- Conservation of certain integrals (linear & quadratic first integrals).

Additionally, ARK methods must also satisfy coupling conditions *between* the methods, to the same accuracy as each elemental method.

ARK Solution Algorithm – Tables

To better understand the workings of an ARK time step, let's consider the ERK/ESDIRK pair, ARK3(2)4L[2]SA [Kennedy & Carpenter, 2001],

| | | | | |
|------|------|-------|------|------|
| 0 | 0 | 0 | 0 | 0 |
| 0.87 | 0.87 | 0 | 0 | 0 |
| 0.6 | 0.53 | 0.07 | 0 | 0 |
| 1.0 | 0.40 | -0.44 | 1.04 | 0 |
| | 0.19 | -0.60 | 0.97 | 0.44 |
| | 0.21 | -0.49 | 0.87 | 0.40 |

| | | | | |
|------|------|-------|------|------|
| 0 | 0 | 0 | 0 | 0 |
| 0.87 | 0.44 | 0.44 | 0 | 0 |
| 0.6 | 0.26 | -0.09 | 0.44 | 0 |
| 1.0 | 0.19 | -0.60 | 0.97 | 0.44 |
| | 0.19 | -0.60 | 0.97 | 0.44 |
| | 0.21 | -0.49 | 0.87 | 0.40 |

ARK Solution Algorithm – Stage 1

To better understand the workings of an ARK time step, let's consider the ERK/ESDIRK pair, ARK3(2)4L[2]SA [Kennedy & Carpenter, 2001],

| | | | | |
|------|------|-------|------|------|
| 0 | 0 | 0 | 0 | 0 |
| 0.87 | 0.87 | 0 | 0 | 0 |
| 0.6 | 0.53 | 0.07 | 0 | 0 |
| 1.0 | 0.40 | -0.44 | 1.04 | 0 |
| | 0.19 | -0.60 | 0.97 | 0.44 |
| | 0.21 | -0.49 | 0.87 | 0.40 |

| | | | | |
|------|------|-------|------|------|
| 0 | 0 | 0 | 0 | 0 |
| 0.87 | 0.44 | 0.44 | 0 | 0 |
| 0.6 | 0.26 | -0.09 | 0.44 | 0 |
| 1.0 | 0.19 | -0.60 | 0.97 | 0.44 |
| | 0.19 | -0.60 | 0.97 | 0.44 |
| | 0.21 | -0.49 | 0.87 | 0.40 |

Stage 1: $z_1 = u_n$,

ARK Solution Algorithm – Stage 2

To better understand the workings of an ARK time step, let's consider the ERK/ESDIRK pair, ARK3(2)4L[2]SA [Kennedy & Carpenter, 2001],

| | | | | | | | | | |
|------|------|-------|------|------|------|------|-------|------|------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.87 | 0.87 | 0 | 0 | 0 | 0.87 | 0.44 | 0.44 | 0 | 0 |
| 0.6 | 0.53 | 0.07 | 0 | 0 | 0.6 | 0.26 | -0.09 | 0.44 | 0 |
| 1.0 | 0.40 | -0.44 | 1.04 | 0 | 1.0 | 0.19 | -0.60 | 0.97 | 0.44 |
| | 0.19 | -0.60 | 0.97 | 0.44 | | 0.19 | -0.60 | 0.97 | 0.44 |
| | 0.21 | -0.49 | 0.87 | 0.40 | | 0.21 | -0.49 | 0.87 | 0.40 |

Stage 1: $z_1 = u_n$,

Stage 2: $z_2 - 0.44hf_I(t_{n,2}, z_2) = u_n + h \left(a_{2,1}^E f_E(t_{n,1}, z_1) + a_{2,1}^I f_I(t_{n,1}, z_1) \right),$

ARK Solution Algorithm – Stage 3

To better understand the workings of an ARK time step, let's consider the ERK/ESDIRK pair, ARK3(2)4L[2]SA [Kennedy & Carpenter, 2001],

| | | | | | | | | | |
|------|------|-------|------|------|------|------|-------|------|------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.87 | 0.87 | 0 | 0 | 0 | 0.87 | 0.44 | 0.44 | 0 | 0 |
| 0.6 | 0.53 | 0.07 | 0 | 0 | 0.6 | 0.26 | -0.09 | 0.44 | 0 |
| 1.0 | 0.40 | -0.44 | 1.04 | 0 | 1.0 | 0.19 | -0.60 | 0.97 | 0.44 |
| | 0.19 | -0.60 | 0.97 | 0.44 | | 0.19 | -0.60 | 0.97 | 0.44 |
| | 0.21 | -0.49 | 0.87 | 0.40 | | 0.21 | -0.49 | 0.87 | 0.40 |

Stage 1: $z_1 = u_n$,

Stage 2: $z_2 - 0.44hf_I(t_{n,2}, z_2) = u_n + h \left(a_{2,1}^E f_E(t_{n,1}, z_1) + a_{2,1}^I f_I(t_{n,1}, z_1) \right)$,

Stage 3: $z_3 - 0.44hf_I(t_{n,3}, z_3) = u_n + h \sum_{j=1}^2 \left(a_{3,j}^E f_E(t_{n,j}, z_j) + a_{3,j}^I f_I(t_{n,j}, z_j) \right)$,

ARK Solution Algorithm – Stage 4

To better understand the workings of an ARK time step, let's consider the ERK/ESDIRK pair, ARK3(2)4L[2]SA [Kennedy & Carpenter, 2001],

| | | | | | | | | | |
|------|------|-------|------|------|------|------|-------|------|------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.87 | 0.87 | 0 | 0 | 0 | 0.87 | 0.44 | 0.44 | 0 | 0 |
| 0.6 | 0.53 | 0.07 | 0 | 0 | 0.6 | 0.26 | -0.09 | 0.44 | 0 |
| 1.0 | 0.40 | -0.44 | 1.04 | 0 | 1.0 | 0.19 | -0.60 | 0.97 | 0.44 |
| | 0.19 | -0.60 | 0.97 | 0.44 | | 0.19 | -0.60 | 0.97 | 0.44 |
| | 0.21 | -0.49 | 0.87 | 0.40 | | 0.21 | -0.49 | 0.87 | 0.40 |

Stage 1: $z_1 = u_n$,

Stage 2: $z_2 - 0.44hf_I(t_{n,2}, z_2) = u_n + h \left(a_{2,1}^E f_E(t_{n,1}, z_1) + a_{2,1}^I f_I(t_{n,1}, z_1) \right)$,

Stage 3: $z_3 - 0.44hf_I(t_{n,3}, z_3) = u_n + h \sum_{j=1}^2 \left(a_{3,j}^E f_E(t_{n,j}, z_j) + a_{3,j}^I f_I(t_{n,j}, z_j) \right)$,

Stage 4: $z_4 - 0.44hf_I(t_{n,4}, z_4) = u_n + h \sum_{j=1}^3 \left(a_{4,j}^E f_E(t_{n,j}, z_j) + a_{4,j}^I f_I(t_{n,j}, z_j) \right)$,

ARK Solution Algorithm – Finish

To better understand the workings of an ARK time step, let's consider the ERK/ESDIRK pair, ARK3(2)4L[2]SA [Kennedy & Carpenter, 2001],

| | | | | | | | | | |
|------|------|-------|------|------|------|------|-------|------|------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.87 | 0.87 | 0 | 0 | 0 | 0.87 | 0.44 | 0.44 | 0 | 0 |
| 0.6 | 0.53 | 0.07 | 0 | 0 | 0.6 | 0.26 | -0.09 | 0.44 | 0 |
| 1.0 | 0.40 | -0.44 | 1.04 | 0 | 1.0 | 0.19 | -0.60 | 0.97 | 0.44 |
| | 0.19 | -0.60 | 0.97 | 0.44 | | 0.19 | -0.60 | 0.97 | 0.44 |
| | 0.21 | -0.49 | 0.87 | 0.40 | | 0.21 | -0.49 | 0.87 | 0.40 |

Stage 1: $z_1 = u_n$,

Stage 2: $z_2 - 0.44hf_I(t_{n,2}, z_2) = u_n + h \left(a_{2,1}^E f_E(t_{n,1}, z_1) + a_{2,1}^I f_I(t_{n,1}, z_1) \right)$,

Stage 3: $z_3 - 0.44hf_I(t_{n,3}, z_3) = u_n + h \sum_{j=1}^2 \left(a_{3,j}^E f_E(t_{n,j}, z_j) + a_{3,j}^I f_I(t_{n,j}, z_j) \right)$,

Stage 4: $z_4 - 0.44hf_I(t_{n,4}, z_4) = u_n + h \sum_{j=1}^3 \left(a_{4,j}^E f_E(t_{n,j}, z_j) + a_{4,j}^I f_I(t_{n,j}, z_j) \right)$,

Finish: $u_{n+1} = u_n + h \sum_{j=1}^4 b_j (f_{E,j} + f_{I,j})$ and $\tilde{u} = u_n + h \sum_{j=1}^4 \tilde{b}_j (f_{E,j} + f_{I,j})$.

The ARKode Library

As a part of the FASTMath SciDAC Institute, we are constructing a library comprising these solvers named *ARKode*, that will be released as a new component solver within SUNDIALS.

- Nearly identical user interface as CVODE, albeit with separate user-specified $f_E(t, y)$ and $f_I(t, y)$ routines.
- Data structure agnostic – as long as the basic vector kernels are supplied this works with your native data structures.
- High-order accurate dense output, allowing efficient interpolation of results between integration steps, and reliable implicit predictors.
- Parameters optimized for iterative solvers and large-scale parallelism.
- Exhaustive suite of example and regression test problems.

The ARKode Library

Key differences between ARKode and CVODE include:

- Support for f_E and/or f_I , allowing adaptive ERK, DIRK or ARK.
- Optional accelerated fixed-point nonlinear solver,
- Optional PCG, FGMRES Krylov solvers,
- Support for non-identity mass matrix, $Mu' = f_E(t, u) + f_I(t, u)$,
- “Hot restart” support for problems with spatial adaptivity.
- “Set routines” allowing complete control over: Butcher table coefficients, time step adaptivity algorithm, temporal error estimation algorithm, implicit predictor algorithm, all internal solver parameters.

Plans:

- Fall 2013 public release; “friendly-user” release available now.
- Future support for partitioned symplectic methods (Hamiltonian systems).

Transitioning Between CVODE and ARKode

Consistent API simplifies experimentation, e.g.:

```
#include <cvode/cvode.h>
#include <cvode/cvode_dense.h>
```

```
/* User-supplied Functions */
static int f(realtype t, N_Vector y,
             N_Vector ydot, void *udata);
static int Jac(long int N, realtype t, N_Vector y,
               N_Vector fy, DlsMat J, void *udata,
               N_Vector t1, N_Vector t2, N_Vector t3);
```

```
int main() {
```

```
...
```

```
void *cvode_mem = CVodeCreate(CV_BDF, CV_NEWTON);
CVodeInit(cvode_mem, f, T0, y);
CVodeSStolerances(cvode_mem, reltol, abstol);
CVDense(cvode_mem, NEQ);
CVDlsSetDenseJacFn(cvode_mem, Jac);
```

```
for (iout=0; iout<Nt; iout++)
    CVode(cvode_mem, tout, y, &t, CV_NORMAL);
```

```
}
```

```
#include <arkode/arkode.h>
#include <arkode/arkode_dense.h>
```

```
/* User-supplied Functions */
static int fe(realtype t, N_Vector y,
              N_Vector ydot, void *udata);
static int fi(realtype t, N_Vector y,
              N_Vector ydot, void *udata);
static int Jac(long int N, realtype t, N_Vector y,
               N_Vector fy, DlsMat J, void *udata,
               N_Vector t1, N_Vector t2, N_Vector t3);
```

```
int main() {
```

```
...
```

```
void *arkode_mem = ARKodeCreate();
ARKodeInit(arkode_mem, fe, fi, T0, y);
ARKodeSStolerances(arkode_mem, reltol, abstol);
ARKDense(arkode_mem, NEQ);
ARKDlsSetDenseJacFn(arkode_mem, Jac);
```

```
for (iout=0; iout<Nt; iout++)
    ARKode(arkode_mem, tout, y, &t, ARK_NORMAL);
```

```
}
```

Outline

- 1 Motivation
- 2 ARK Methods
- 3 Example Results**
- 4 Conclusions

Hyperstability Test Problem

We also ran the same unstable test problem that Carol showed:

$$\partial_t u = 100u - 400v, \quad u(0) = 2,$$

$$\partial_t v = 100u + 100v, \quad v(0) = 2,$$

for $t \in [0, 0.25]$, having analytical solutions

$$u(t) = e^{100t} [2 \cos(200t) - 4 \sin(200t)],$$

$$v(t) = e^{100t} [2 \cos(200t) + \sin(200t)].$$

All ARKode methods correctly tracked the instability, providing at least one significant digit when using $rtol = atol = 10^{-2}$. For each method class/order, our adaptivity strategy required the following numbers of steps:

- DIRK: 2 (3017), 3 (205), 4 (205), 5 (50)
- ERK: 2 (2998), 3 (289), 4 (291), 5 (76), 6 (85)
- ARK: 3 (244), 4 (114), 5 (95)

PDE Cosmology Model Problem – Radiating Ionization Front

Find $u(\mathbf{x}, t)$, $v(\mathbf{x}, t)$ s.t. for $(\mathbf{x}, t) \in [-1, 1]^3 \times [0, 1]$,

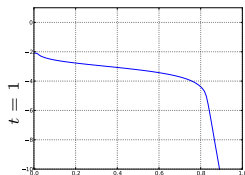
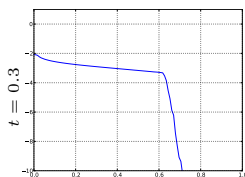
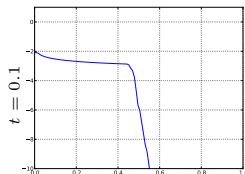
$$\partial_t u = \nabla \cdot (\beta u) + \nabla \cdot (\mu \nabla u) + f_u(u, v),$$

$$\partial_t v = \nabla \cdot (\beta v) + f_v(u, v),$$

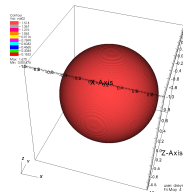
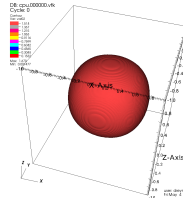
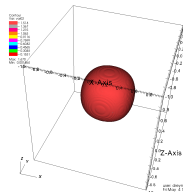
Where

- $u(\mathbf{x}, 0) = 10^{-8}$, $v(\mathbf{x}, 0) = 1.67$.
- $\nabla u \cdot \mathbf{n}|_{\partial\Omega} = 0$, $\nabla v \cdot \mathbf{n}|_{\partial\Omega} = 0$.
- $f_u(u, v) = 25000 \delta_0(\mathbf{x}) - 1800000 uv$.
- $f_v(u, v) = a(c - v)^2 - bv(c - v) + 567000 uv$.
- $\beta(u, v, \mathbf{x}) = -\frac{\alpha u(c - v)}{\|\mathbf{x}\|} \mathbf{x}$, $\alpha = \{10, 100, 250\}$,
- $\mu(v) = \frac{100}{v}$,
- $a = 2.445$, $b = 0.01118$, $c = 1.673$.

$\log_{10}(u)$ Profiles



Ionized region



Radiating Ionization Test Solvers

Discretizations:

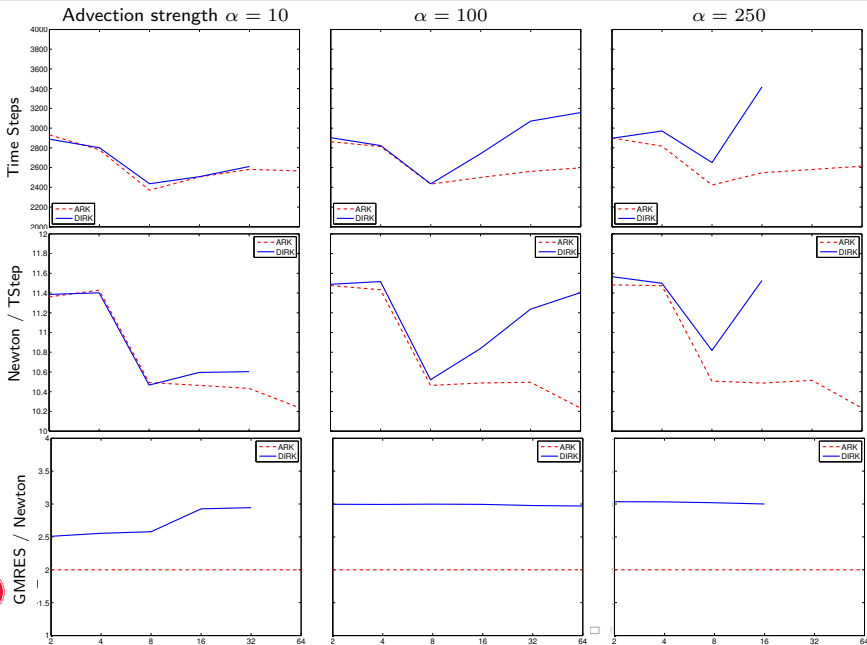
- FV grid of size $N_x \times N_y \times N_z$, with a fixed $32 \times 32 \times 32$ grid per MPI task.
- Process sizes: $2(2 \times 1 \times 1)$, $4(2 \times 2 \times 1)$, $8(2 \times 2 \times 2)$, $16(4 \times 2 \times 2)$, $32(4 \times 4 \times 2)$, $64(4 \times 4 \times 4)$.
- $\nabla \cdot (\mu \nabla u)$ discretized using 2nd-order centered differences.
- $\nabla \cdot (\beta u)$ discretized using CD or $\mathcal{O}(\Delta x)$ upwind, depending on $\|\beta\| > 10^{-3}$.
- Time: $\mathcal{O}(\Delta t^4)$ ARK and DIRK methods.

Solvers:

- Nonlinear: inexact Newton, `maxit` = 5.
- Linear: matrix-free GMRES, `maxit` = 5.
- Precond: AMG [HYPRE], Rx-Diff only (no advection).
- PID controller [Kennedy & Carpenter, 2001], with error estimate

$$e = \left(\frac{1}{N} \sum_i \left(\frac{u_i - \tilde{u}_i}{r_{tol} u_i + a_{tol}} \right)^2 \right)^{1/2}, \quad r_{tol} = 10^{-3}, \quad a_{tol} = 10^{-9}.$$

Radiating Ionization Results – Iteration Weak Scaling



Outline

- 1 Motivation
- 2 ARK Methods
- 3 Example Results
- 4 Conclusions**

Conclusions

ARK methods allow accurate/stable splitting of multi-rate problems:

- IMEX couplings are properly handled,
- No Dahlquist barrier – high accuracy and stability simultaneously possible.

RK under-pinnings allow robust and theoretically rigorous methods for error estimation and time adaptivity.

Works well with spatial adaptivity (due to one-step approach), with tunings for vector resizing and finite element discretizations.

Flexible infrastructure:

- Allows adaptive ERK or IRK methods alone,
- Enables domain-specific knowledge into the formulation.

“Convenient” preconditioners need not be chastised, since many solver difficulties (non-differentiability, nonlinearity, complex discretizations) often lie in slow components that can be treated explicitly.



Thanks & Acknowledgements

Collaborators/Students:

- Carol S. Woodward [LLNL]
- Alan C. Hindmarsh [LLNL]
- David J. Gardner [SMU, PhD]
- Jean M. Sexton [SMU, MS]



Current Grant/Computing Support:

- DOE SciDAC & INCITE Programs
- NSF AST & XSEDE Programs
- DOD DURIP Program
- SMU Center for Scientific Computation



Software:

- ARKode – <http://faculty.smu.edu/reynolds/arkode>
- SUNDIALS – <https://computation.llnl.gov/casc/sundials>
- Enzo (cosmology) – <http://enzo-project.org>



Outline

5 Extra Slides



First-Order Splittings

Denote $S_i(h, u(t_n))$ as a solver for the component $\partial_t u = f_i(t, u)$ over a time step $t_n \rightarrow t_n + h \equiv t_{n+1}$, with initial condition $u(t_n)$.

To evolve $u(t_n) \rightarrow u(t_{n+1})$, we can use different solvers at the same h ,

$$\begin{aligned}\hat{u} &= S_1(h, u(t_n)), \\ u(t_{n+1}) &= S_2(h, \hat{u}),\end{aligned}$$

or we may subcycle time steps for individual components,

$$\begin{aligned}\hat{u}_{j+1} &= S_1\left(\frac{h}{m}, \hat{u}_j\right), \quad j = 0, \dots, m, \quad \hat{u}_0 = u(t_n), \\ u(t_{n+1}) &= S_2(h, \hat{u}_m),\end{aligned}$$

Unless the S_i commute [i.e. $S_1(h, S_2(h, u)) = S_2(h, S_1(h, u))$] or the splitting is symmetric, these methods are at best $O(h)$ accurate (*no matter the accuracy of the individual solvers*).

Fractional Step (Strang) Splitting [Strang 1968]

“Strang splitting” attempts to achieve a higher-order method using these separate component solvers, through manually symmetrizing the operator:

$$\hat{u}_1 = S_1 \left(\frac{h}{2}, u(t_n) \right),$$

$$\hat{u}_2 = S_2 (h, \hat{u}_1),$$

$$u(t_{n+1}) = S_1 \left(\frac{h}{2}, \hat{u}_2 \right).$$

This approach is $O(h^2)$ as long as each S_i is $O(h^2)$.

However:

- This asymptotic accuracy may not be achieved until h is very small, since error terms are dominated by inter-process interactions [Ropp, Shadid,& Ober 2005].
- Numerical stability isn't guaranteed *even if h is stable for each component* [Estep et al., 2007].

Operator-Splitting Issues – Accuracy [Ropp, Shadid, & Ober 2005]

Coupled systems can admit destabilizing modes not present in either component, due to *numerical resonance instabilities* [Grubmüller 1991].

Brusselator Example (Reaction-Diffusion):

$$\partial_t T = \frac{1}{40} \nabla^2 T + 0.6 - 3T + T^2 C,$$

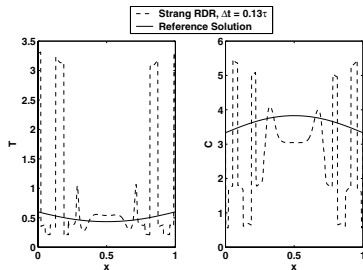
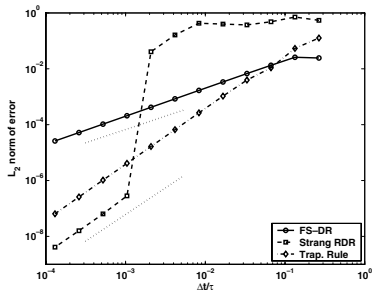
$$\partial_t C = \frac{1}{40} \nabla^2 C + 2T - T^2 C,$$

Three solvers:

- (a) Basic split: **D** (trap.) then **R** (subcycled BDF).
- (b) Strang: $\frac{h}{2}\mathbf{R}$, $h\mathbf{D}$, $\frac{h}{2}\mathbf{R}$,
- (c) Fully implicit trapezoidal rule,

Results:

- (a) is stable but inaccurate for all tests;
- (b) unusable until h is “small enough”.



Operator Splitting Issues – Accuracy [Estep 2007]

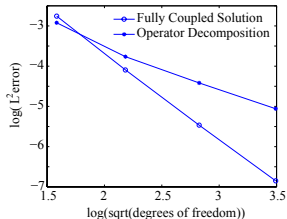
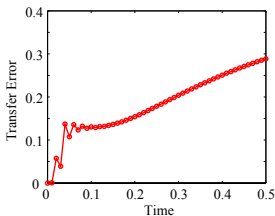
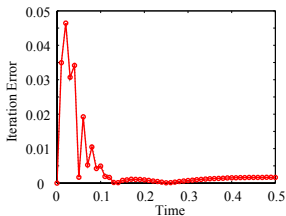
Consider $\Omega = \Omega_1 \cup \Omega_2$ where the subdomains share a boundary $\Gamma = \partial\Omega_1 \cap \partial\Omega_2$:

$$\partial_t u_1 = \nabla^2 u_1, \quad x \in \Omega_1, \quad \partial_t u_2 = \frac{1}{2} \nabla^2 u_2, \quad x \in \Omega_2,$$

$$u_1 = u_2, \quad \nabla u_1 \cdot n = \nabla u_2 \cdot n, \quad \text{for } x \in \Gamma.$$

Solved using one Gauss-Seidel iteration: S_1 on Ω_1 , then S_2 on Ω_2 (both trapezoidal). Errors from not iterating to convergence, and from error transfer between subdomains.

Using adjoints, they measured these errors separately:



- Error from incomplete iteration decreased with time.
- Transfer error accumulated and became dominant with time.
- While each S_i was $O(h^2)$, the coupled method was only $O(h)$.

Operator-Splitting Issues – Stability [Estep et al., 2007]

Second Reaction-Diffusion Example (split subcycling; exact solvers):

$$\partial_t u = u^2 - \lambda u, \quad u(0) = u_0, \quad t > 0.$$

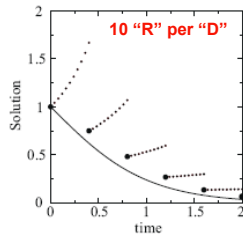
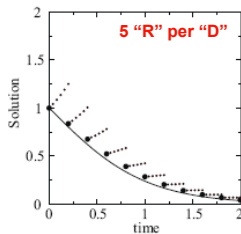
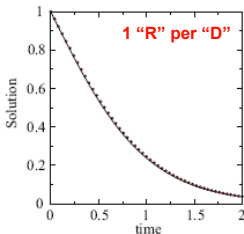
Phase 1 (R): $\partial_t u_r = u_r^2, \quad u_r(t_n) = u_n, \quad t \in [t_n, t_{n+1}],$

Phase 2 (D): $\partial_t u_d = -\lambda u_d, \quad u_d(t_n) = u_r(t_{n+1}), \quad t \in [t_n, t_{n+1}].$

True solution, $u(t) = \frac{u_0 e^{-\lambda t}}{1 + \frac{u_0}{\lambda} (e^{-\lambda t} - 1)},$ is well-defined $\forall t$ if $\lambda > u_0$.

Split solution, $u(t_{n+1}) = \frac{u(t_n) e^{-\lambda h}}{1 - u(t_n) h},$ can blow up in finite time.

Results using 50
time steps, with
varying amounts
of subcycling.



Time Step Selection

Stability:

If IRK portion is A -stable, linear instability can only arise due to the ERK, so time steps must satisfy the explicit stability restriction,

$$h \leq h_{exp}.$$

Accuracy:

For an order q method with order p embedding (typically $q = p + 1$),

$$\begin{aligned} \|u(t_{n+1}) - u_{n+1}\| &\leq \|\tilde{u} - u_{n+1}\| + \|u(t_{n+1}) - \tilde{u}\| \\ &\leq \|\tilde{u} - u_{n+1}\| + \mathcal{O}(h^{p+1}). \end{aligned}$$

Hence h adaptivity may utilize $\|\tilde{u} - u_{n+1}\|$ for error estimation, e.g.

$$h_{n+1} = c h_n \left(\frac{r_{tol}}{\|\tilde{u} - u_{n+1}\|} \right)^{1/(p+1)}$$

In practice, a variety of adaptivity methods may be used, and are often based on control-theoretic techniques (e.g. I, PI, PID) [Gustafsson 1991 & 1994; Söderlind 2003].

Implicit Predictors and Dense Output

Accurate initial guesses are critical for nonlinear solver efficiency & robustness.

After each successful step $t_{n-1} \rightarrow t_n$ we have full-order versions of the solutions and right-hand sides $\{u_{n-1}, u_n, f_{n-1}, f_n\}$. These generate a unique cubic Hermite interpolating polynomial $u_d(t)$, for interpolating intermediate solutions in $[t_{n-1}, t_n]$ or predicting subsequent stage solutions.

- For higher-order $u_d(t)$, a “bootstrapping” approach may be used. This has diminishing returns, so output accuracy capped at $\mathcal{O}(h^5)$.
- Coefficients are determined *a priori* for each desired output order.
- Extrapolation accuracy rapidly diminishes for $t > t_n$, so predictors are most helpful for early stages of next step.
- Later stages in a step z_i may instead be predicted by previous stages, z_1, \dots, z_{i-1} .